

ECO quick start 07 - Creating an ASP.NET application

Table of Contents

Overview	1
Prerequisites and goals	2
Before you start...	3
Web auto forms	4
State machine operation	6
New ECO ASP.NET components	8
Logical flow	10
Creating a web form	12
Confirming deletion	13
Validating user input	15
Exercise for the reader	19
Passing objects between page requests	20
Editing a single object	22
Databinding a dropdown list	26
Master / detail	30
Authorization	32
Role based authorization	35
Accessing the current user	37
EcoSpace retrieval strategies	38
Pooling EcoSpaces	38
Session EcoSpaces	39

Shopping cart	41
Preliminary changes	41
Adding orders to the model	42
The shopping cart control	43
Show all available articles	46
Summary	49
Index	a

1 Overview

ECO introduces two new components for use in ASP.NET applications. These components work with the new ASP.NET DataSourceControl approach for creating websites. Although it is possible to use standard ECO handles in ASP.NET applications it is recommended that these new components (EcoDataSource, EcoSpaceManager) are used instead, and therefore these are the only components that will be covered in this tutorial.

2 Prerequisites and goals

Prerequisites

To successfully follow this document the user should have read the preceding articles in this series and have a copy of the model created.

Goals

By the end of this document you will be able to

- Create ECO ASP.NET applications.
- Use instances of ECO modeled classes with standard ASP.NET controls such as DataGridView.
- Create a master/detail web UI.
- Bind drop down lists of objects to the property of a single object.
- Validate objects.
- Pass objects references across page requests.
- Manage EcoSpace life time and pooling options.
- Work with multiple EcoSpace instances and session settings.

3 Before you start...

The `persistMapperXml1.FileName` property on `PersistenceMapperProvider` is set to "data.xml". As this is a relative path the file will be created relative to the project that creates the `PersistenceMapperProvider`, so effectively the expected location of the file will change depending on whether you run the WinForm application or Web application.

Ordinarily you would use a proper data storage in a situation like this, but to avoid having to choose a specific RDBMS for these QuickStart articles the XML persistence will continue to be used. In order to achieve this a small change must be made to the `PersistenceMapperProvider` component so that the data.xml file is always located in a predictable location.

View the source for the `PersistenceMapperProvider` and change the source as follows:

```
public QuickStartPMP(): base()
{
    InitializeComponent();
    //Use a predictable file path
    SetXmlFileName();
}

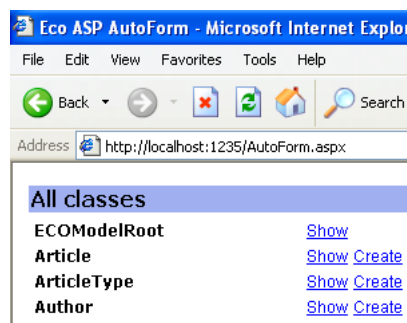
private void SetXmlFileName()
{
    string xmlFileName =
Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);
    xmlFileName = System.IO.Path.Combine(xmlFileName, "CapableObjects\\ECO Quick
Start\\SharedData");
    System.IO.Directory.CreateDirectory(xmlFileName);
    xmlFileName = System.IO.Path.Combine(xmlFileName, "data.xml");
    persistMapperXml1.FileName = xmlFileName;
}
```

If you have chosen to use an RDBMS as your data storage there is no need to perform these steps.

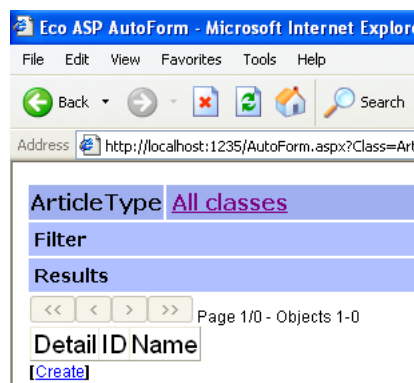
4 Web auto forms

If you look in your QuickStart.ASP project you will see a web form named "AutoForm". Web auto forms are the ASP equivalent of the prototype tool in EcoModeler; they allow you to quickly test your model in a web environment by browsing classes in the model, creating, updating, deleting instances, and linking / unlinking objects via their associations.

Ensure the ASP project is the default by right-clicking it and selecting "Set As SetUp Project", then ensure the AutoForm is the default page by right-clicking it and selecting "Set As Start Page". Now run the project.



1. On the auto form you will see a list of classes in the model. To start click "Show" hyperlink next to ArticleType. You will now be presented with a form like so:



Note: If you performed the steps in QuickStart 6 (Making the EcoSpace persistent) you will see data. If not you will see no data because the path to the data file in persistenceMapperXml1.FileName is relative to the running application. If you intend to use the same XML file for both WinForm and Web app projects you will need to change this path. Obviously it is not recommended that you use XML for a released application.

2. Click the "Create" hyperlink to create a new instance of ArticleType.

[ArticleType](#)

Attributes

ID

Name

- Enter "WinForm" as the name of the ArticleType and click the "Submit Changes" button to save.
- Now click the "ArticleType" hyperlink at the top left of the form to return to the list of ArticleType instances.
- Click "Create" again and create another article type named "ASP .NET" and save it. Note how you cannot change the ID, this text box has been made read only because it is auto-assigned by the database.
- Now click the "All classes" hyperlink at the top right of the form. Create two new Author instances.

Author [All classes](#)

Filter

Results

<< < > >> Page 1/1 - Objects 1-2

Detail	EmailAddress	FirstName	ID	LastName	Password	Salutation	FullName	AutoPublishArticles
Detail	me@home.com	John	1	Smith	secret	Mr	Mr John Smith	True
Detail	him@home.com	Fred	3	Smith	secret	Mr	Mr Fred Smith	False

[\[Create\]](#)

- Now click the "Detail" hyperlink for the first author to view the author's details.
- At the bottom of the form is a list of roles (association ends.) In this list you will see "Articles", click the "Add New Article" link.

[Article](#) **Article** [All classes](#)

Attributes -

Body

Description

ID

Title

Summary

ReasonRejected

ReviewState

MainState

Roles -

ArticleType [\[Link\]](#)

Author [Mr John Smith](#) [\[Unlink\]](#) [\[Link\]](#)

Operations -

State Machines -

Article.Authoring

- Enter some information and click the "Submit Changes" button.
- Next to the ArticleType you will see a hyper link entitled "Link", this allows you to link the Article to an ArticleType, click this link now.

ArticleType All classes

Filter

Results

<< < > >>

Page 1/1 - Objects 1-2

Detail ID	Name	Link
<u>Detail 1</u>	WinForm	<u>Link</u>
<u>Detail 3</u>	ASP .NET	<u>Link</u>

[Create]

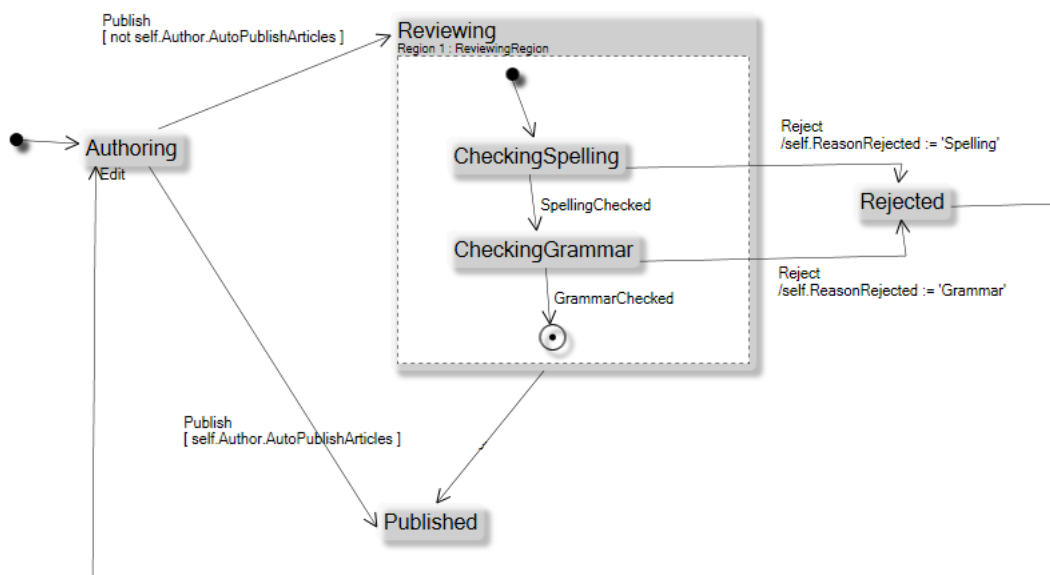
11. You will now be presented with a list of ArticleType instances created earlier. Next to each item there is a hyperlink entitled "Link". Click one of these links.
12. You will now be returned to the Article you were editing, the ArticleType has now been set. It is also possible to unlink objects by clicking the "Unlink" hyperlink.

Roles	
ArticleType	WinForm [Unlink] [Link]
Author	Mr John Smith [Unlink] [Link]

Repeat these steps to create an article for the second Author instance.

4.1 State machine operation

At the bottom of the auto form for Article you may have noticed there is a section entitled "State Machines". Within this section there is a single button entitled "Publish". If you think back to the 4th article in this series (Creating a state machine) you will remember the state machine diagram for the Article class and how it behaved during prototyping.



1. Select the Author with AutoPublishArticles set to True.

- At the bottom of the Author auto form you will see a Roles section entitled "Articles", click the "Detail" hyperlink next to the article in the list.
- At the bottom of the Article auto form you will see the State Machine section and a single button entitled "Publish".

State Machines	
Article.Authoring	<input type="button" value="Publish"/>

- Click the "Publish" button, the state of the Article will change from Authoring to Published, and the "Publish" button will be replaced with an "Edit" button allowing you to put the Article state back to Authoring.
- Now click the "All classes" hyperlink and then the "Show" hyperlink next to the Author class.
- This time click the "Detail" hyperlink next to the Author that has AutoPublishArticles set to False, then click the "Detail" hyperlink next to the Author's only Article.
- Scroll to the bottom of the page to where the State Machine section is located and click the "Publish" button.

State Machines	
Article.Reviewing	<input type="button" value="Edit"/>
Reviewing.CheckingSpelling	<input type="button" value="SpellingChecked"/>
	<input type="button" value="Reject"/>

- The state machine will now enter the Reviewing state, which according to the state machine diagram is a composite state containing two more states, "CheckingSpelling" and "CheckingGrammar". As you can see the generated HTML accounts for embedded regions too.
- Click the "SpellingChecked" button and it will be replaced with a button entitled "GrammarChecked".
- Now click the "Reject" button and the state will be set to "Rejected", in addition the ReasonRejected text box will read "Grammar".

Article	Article	All classes
Attributes		
Body	Using auto forms to pro	
Description	This article describes h	
ID	21	
Title	ECO Web AutoForms	
Summary	ECO Web AutoForms br	
ReasonRejected	<input type="text" value="Grammar"/>	
ReviewState	<input type="text" value="CheckingGrammar"/>	
MainState	<input type="text" value="Rejected"/>	
	<input type="button" value="Submit Changes"/>	<input type="button" value="Delete Instance"/>
Roles		
ArticleType	[Link]	
Author	Mr Fred Smith [Unlink] [Link]	
Operations		
State Machines		
Article.Rejected	<input type="button" value="Edit"/>	

The Article will now be in a state where the Author can see that their article has been rejected and why, they can then put the article back into the Authoring state by clicking the "Edit" button.

5 New ECO ASP.NET components

In ECO IV the interaction between objects in an EcoSpace and the web UI controls is handled by two new components.

EcoSpaceManager

This component makes an EcoSpace instance available during the rendering of a web form, at the end of the rendering process this component will ensure that the correct action is taken to release the EcoSpace instance. This behavior depends upon a number of settings on the EcoSpaceManager and also in the web.config file. Important properties of this component are:

- ID: The unique name of this component.
- EcoSpaceTypeName: The drop down list will present a list of EcoSpace types available in your project. In this case there will be only one, but it is possible to have different types of EcoSpace in the same project with each having its own model. When the EcoSpace property of this component is read it will ensure that an EcoSpace of the specified type is returned.
- SessionStateKey: The first time the EcoSpace property is read this component will first see if there is a key in the ASP.NET Session with the given name. If there is then the EcoSpace held in the session with this key will be returned.
- SessionStateMode: This setting specifies whether or not to place the EcoSpace instance into the ASP.NET session at the end of the page render.
 - Always: The EcoSpace instance will always be put into the session at the end of the page render.
 - IfDirty: The EcoSpace instance will be put into the session at the end of the page render only if it has dirty objects (modified persistent objects which have not been saved to the data storage.)
 - Never: The EcoSpace instance will not be put into the session, it will either be disposed of or placed back into a pool. If there are dirty objects in the EcoSpace at the end of the page render an exception will be thrown.

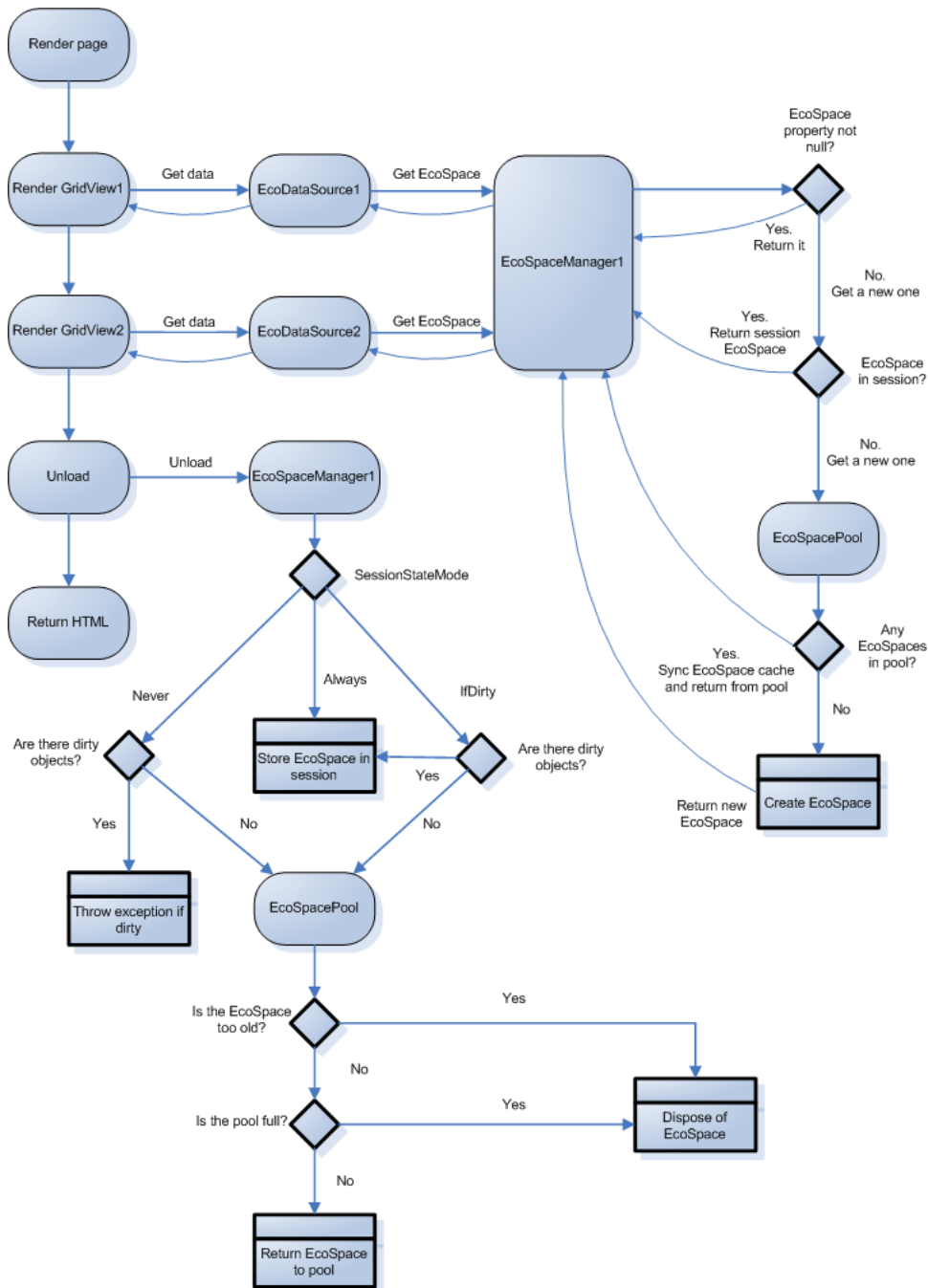
EcoDataSource

This component is responsible for controlling interaction between objects in the EcoSpace and the web UI controls. It obtains its EcoSpace instance from an EcoSpaceManager. Important properties of this component are:

- AddDefaultProperties: If set to True then columns will be created automatically at runtime to represent the data made available by the Expression and PsExpression properties. If a column has been added at design time with a conflicting name then it will take precedence.
- AddExternalId: If set to True a column will be added named "ExternalId" which provides read only access to a string which can be used to uniquely identify an instance of a modeled business class. This option is only valid if the result of evaluating the expressions results in a single business object or a collection of business objects.
- DeleteMode: Specifies the action to take when the user clicks Delete on a new web control such as GridView:
 - Disallow: No action will be performed.
 - Delete: Deletes the selected object, this option is only valid if evaluating the Expression and PsExpression properties returns a single object or a collection of objects.
 - Unlink: The selected object will be removed from the list, this option is only valid if evaluating the Expression property returns a multi role, such as self.purchaseOrder.lines.
- EcoSpaceManagerID: Identifies which EcoSpaceManager to obtain an EcoSpace instance from during rendering of the web form. At design time this is used to identify the EcoSpace type and therefore the model information.
- Expression / PsExpression: These expressions are executed automatically during rendering of the web form. First the PsExpression is executed, and then the Expression is evaluated. If there is no PsExpression then the Expression is evaluated against the EcoSpace, otherwise it is evaluated against the result of executing the PsExpression.

- **InsertMode:** Specifies the action to take when `EcoDataSource.Insert` is executed. You should use the `Insert` so that ASP.NET is aware the binding list has changed.
 - **Disallow:** No action will be taken.
 - **AddNew:** Adds a new instance of the correct type to the collection. This is only valid if evaluating the expressions results in a collection of business objects, either `AllInstances` of a class or a multi role member of a business object.
 - **CreateNew:** Always creates a new instance of the correct class type.
- **ManualUpdate:** If set to `False` then any changes made via `Insert` / `Delete` / `Edit` will be updated to the data storage immediately. If set to `True` then no update will be performed, in which case the developer must call `UpdateDatabase` or ensure that the `EcoSpaceManager.SessionStateMode` is set appropriately to allow the page rendering process to finish with dirty objects.
- **MemberVisibility:** Specifies the maximum exposure level of properties that will be added automatically when `AddDefaultProperties` is set to `True`.
- **NullRowMode:** When the resulting element of evaluating the expressions is a collection this property specifies where to add an empty row representing `NULL`. This can be useful for example when you wish to present a list of objects for the user to choose a value from and you wish to allow them the option of selecting nothing.
 - **None:** No additional row will be added to the collection.
 - **First:** An additional row will be added to the top of the collection.
 - **Last:** An additional row will be added to the bottom of the collection.
 - **IfEmpty:** An additional row will only be added to the collection if the collection is empty.
- **SupportSorting:** If set to `True` then web UI controls that implement sorting will enable their sorting functionality.
- **ValidateRowOnInsert / ValidateRowOnUpdate :** If set to `True` the `ValidatingRow` event will be executed, providing you with the option to validate the changes to the object before they are applied permanently to the local `EcoSpace` cache.

5.1 Logical flow



This diagram illustrates the life cycle of an ASPX page request from the point of view of the ECO ASP .NET components.

When web UI controls are data bound to an `EcoDataSource` they will attempt to read the data values of the component. If the data has not already been determined the `EcoDataSource` will execute its expressions like so

1. Set the evaluation context to null (the EcoSpace.)
2. If there is a PsExpression then execute it and set the evaluation context to the result of the execution.
3. If there is an Expression then evaluate it in memory and set the context to the result of the evaluation.
4. Set the resulting element (to data bind to) to the result context.

In order to evaluate expressions etc the EcoDataSource requires an instance of an EcoSpace. To obtain this instance it will retrieve it from the EcoSpaceManager's EcoSpace property. To return an EcoSpace the EcoSpaceManager will go through the following process

1. If the EcoSpace property is not null then return the value immediately. The property will not be null if the EcoSpace property has already been read during the page rendering process.
2. If SessionState is not set to Never, and there is an EcoSpace stored in the session under the specified key (the value entered into SessionStateKey) then return this value.
3. If there are any EcoSpaces available in the pool (see the Web.config file for setting the pool size) then
 1. Get an instance from the pool.
 2. If there is a SyncHandler on the PersistenceMapperProvider then get all changes made by other users and sync them into the EcoSpace's cache.
 3. Return the fetched EcoSpace.
4. Otherwise create a new instance of the specified EcoSpaceTypeName will be created and returned.

Subsequent access to the EcoSpaceManager.EcoSpace property will return the instance last returned. This value is held until the end of the page rendering process when the EcoSpaceManager is unloaded. At this point the EcoSpace manager will relinquish its reference to the EcoSpace and go through the following process

1. If the SessionState is set to Never
 1. If the EcoSpace contains dirty objects then throw an exception.
 2. Otherwise return it to the pool.
2. If the SessionState is Always
 1. Put the EcoSpace into the session using the value in SessionStateKey as the key.
3. If the SessionState is IfDirty
 1. If there are dirty objects in the EcoSpace then put it into the session
 2. Otherwise return it to the pool.

When returning an EcoSpace to the pool the EcoSpace will only be added back if the following conditions are true

1. The number of items in the pool is less than the maximum pool size in Web.config.
2. The amount of time elapsed since the EcoSpace instance was created is less than the max age setting in web.config.

6 Creating a web form

Using the new ECO components for web forms means that there is no need for any kind of specialized "ECO web page". Simply drop these new components onto any existing web form and you are able to access instances of your modeled classes. As an exercise we'll recreate an Eco WebForm completely from scratch. This means that it is easy to make any existing web form able to support ECO, or to use ECO on Web Form Content pages too.

1. Delete WebForm1.aspx
2. Create a new Master Page to the project.
3. Add a new Web Content Form to the project named "Authors".
4. Select the master page.
5. Set the new content page as the project's start page by right-clicking it and selecting "Set As Start Page".

The following steps will add the ECO components to the content form to enable it to access instances of your modeled classes. These same steps can be used for normal web forms and also existing web forms.

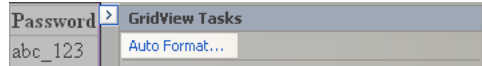
6. Add an EcoSpaceManager to the content form.
 - Set its EcoSpaceTypeName to QuickStartEcoSpace.
 - Note that the SessionStateKey will also default to the same value.
7. Add an EcoDataSource and set it's EcoSpaceManagerID to EcoSpaceManager1.
 - Set its ID to AuthorDataSource.
 - Set its EcoSpaceManagerID to EcoSpaceManager1.
 - Use the [...] editor on the Expression property and set its Expression to

```
Author.allInstances
```

Now the EcoDataSource (AuthorDataSource) has a unique identity, and a way of obtaining an EcoSpace and Element (business objects) at runtime.

8. From the Data section of the component toolbox add a GridView control to the form.
 - Set its ID property to AuthorGridView.
 - Set its DataSourceID to AuthorDataSource.
 - Use the [...] editor on the Columns property to remove all columns except for the following
 - AutoPublishArticles
 - Salutation
 - FirstName
 - LastName
 - FullName
 - EmailAddress
 - Password

- Rearrange the columns into the above order.
- Click each column in turn and give it an appropriate HeaderText property.
- Ensure that the ReadOnly property for the FullName column is True.
- Click the Tasks button at the top right of the GridView.



9. Select the Auto Format option on the popup window and choose a nice color scheme.

10. Before closing the tasks window also set Enable Sorting to True.

11. Finally on the AuthorGridView set both AutoGenerateEditButton and AutoGenerateDeleteButton to True.

Now run the application again and you should see data from the database that was added earlier. If there are no instances of Author then you will see nothing, this is because the GridView control does not output HTML if there is no data.

Click the Edit column next to an Author and the UI will change so that you may edit the information and then either click Update to keep your changes or Cancel to revert.

Welcome to my ECO driven website

<u>Auto publish</u>	<u>Salutation</u>	<u>First name</u>	<u>Last name</u>	<u>Full name</u>	<u>Email address</u>	<u>Password</u>
Update Cancel <input type="checkbox"/>	Mr	John	Smith	Mr John Smith	me@home.com	secret
Edit Delete <input checked="" type="checkbox"/>	Mr	Fred	Smith	Mr Fred Smith	him@home.com	secret

Adding new authors

1. Add a Button to the form beneath the AuthorGridView.
2. Set its ID to NewAuthorButton.
3. Set its Text property to New author.
4. Set the button's Click event code as follows

```
protected void NewAuthorButton_Click(object sender, EventArgs e)
{
    AuthorDataSource.Insert();
}
```

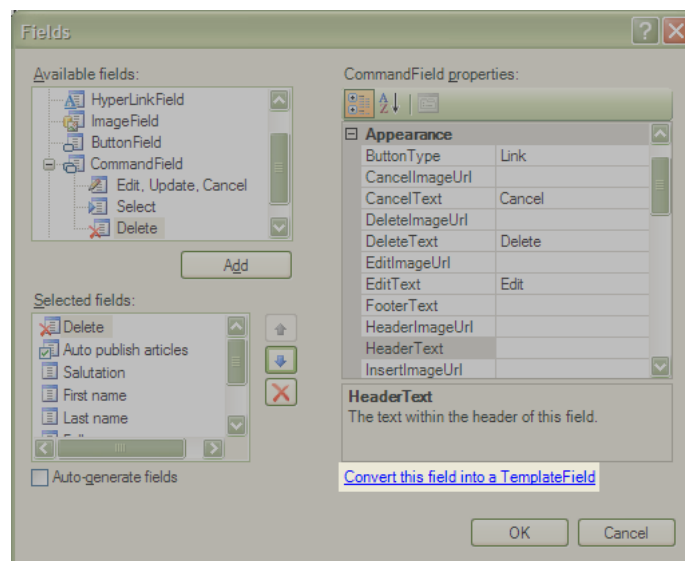
Re-run the application and click the "New author" button, you will see a new row appear. Click Delete to delete the new Author. Note that the Author is deleted immediately without confirmation, this will be addressed next.

6.1 Confirming deletion

Clicking the Delete link next to a row will delete the Author immediately. This is not ideal as it is quite easy to accidentally click the wrong link on a web page. A JavaScript confirmation box will now be added to ensure that the user really does wish

to delete the row.

1. On the AuthorGridView set AutoGenerateDeleteButton to False.
2. Use the [...] editor on its Columns property.
3. In the Available Fields list at the top left of the form expand the tree node entitled "CommandField".
4. Select the child node entitled "Delete" and then click the Add button.
5. Use the arrows to rearrange the Delete column so that it is at the top of the "Selected fields" list, this will ensure it appears along side the Edit button.
6. With the Delete button selected click the link entitled "Convert this field into a TemplateField" at the bottom right of the form.



7. Click the OK button.
8. Now switch the editor to the Source view.
9. Look for a node named LinkButton in the following path

```
<body>
  <form>
    <asp:GridView>
      <Columns>
        <asp:TemplateField>
          <ItemTemplate>
            <asp:LinkButton>
```

10. Change the LinkButton's ID to DeleteAuthorLinkButton.
11. Now switch back to the Design view and in the AuthorGridView.RowDataBound event add the following code.

```
protected void AuthorGridView_RowDataBound(object sender, GridViewRowEventArgs e)
{
    LinkButton DeleteButton;
    if (e.Row.RowType == DataControlRowType.DataRow)
    {

```

```

DeleteButton = e.Row.FindControl("DeleteAuthorLinkButton") as LinkButton;
DeleteButton.OnClientClick = "return confirm('Are you sure you want to delete this
author?');";
}
}

```

Now run the application again and click the Delete link. A dialog box will appear asking you if you are sure you wish to delete the author. Clicking Cancel will abort the deletion, whereas clicking OK will continue as before.

Welcome to my ECO driven website

<u>Edit</u>	<u>Delete</u>	<u>Auto publish</u>	<u>Salutation</u>	<u>First name</u>	<u>Last name</u>	<u>Full name</u>	<u>Email address</u>	<u>Password</u>
Edit	Delete	<input type="checkbox"/>	Mr	John	Smith	Mr John Smith	me@home.com	secret
Edit	Delete	<input checked="" type="checkbox"/>	Mr	Fred	Smith	Mr Fred Smith	him@home.com	secret
Edit	Delete	<input type="checkbox"/>						

The page at http://localhost:1235 says:

Are you sure you want to delete this author?

6.2 Validating user input

This next topic will implement validation for user input. Before allowing the EcoSpace to update the data storage the OCL constraints entered against the class in the model will be evaluated. For each broken constraint an error message will be displayed on the form. If there are any broken constraints the update will not be permitted.

Creating a static validator class

1. Create a new class by right-clicking on "QuickStart.Asp" and selecting the menu item Add->New Item and selecting "Class".
2. Name the class StaticValidator.
3. The implementation of the class is as follows

```

public class StaticValidator : object, IValidator
{
    private readonly string errorMessage;

    public StaticValidator(string errorMessage)
    {
        this.errorMessage = errorMessage;
    }

    string IValidator.ErrorMessage
    {
        get { return errorMessage; }
        set { }
    }

    bool IValidator.IsValid
    {

```

```

        get { return false; }
        set { }
    }

    void IValidator.Validate()
    {
    }
}

```

Using this class it is possible to add error messages which will appear in the ASP.NET ValidationSummary control with the following code

```
Page.Validators.Add(new StaticValidator("This is an error message"));
```

Creating a PageHelper class

As validation is likely to be performed on multiple pages it makes sense that the code to perform the validation should be in a utility class.

1. Add DroopyEyes.Eco.Validation to the References of your QuickStart.ASP project. This file can be found in your {Program Files}\CapableObjects\ECO\5.0\Bin folder.
2. Create a new class.
3. Name it PageHelper.
4. The implementation of the class is as follows:

```

using System.Collections.Generic;
using System.Web.UI;
using Eco.ObjectRepresentation;
using DroopyEyes.EcoExtensions.Validation;

namespace QuickStart
{
    public static class PageHelper
    {
        public static bool ValidatePage(Page currentPage, IObjectInstance affectedObject)
        {
            //Create a list to hold the constraints
            List<IConstraint> constraints = new List<IConstraint>();

            //Use the DroopyEyes ModeledConstraintProvider to get constraints from the
model
            ModeledConstraintProvider constraintProvider = new
ModeledConstraintProvider();

            bool tempResult = true;
            //Get all constraints for the object
            constraintProvider.GetConstraintsForObject(affectedObject, constraints);

            //Create a StaticValidator for each broken constraint
            foreach (IConstraint c in constraints)
            {
                if (!c.IsValid)
                {
                    StaticValidator v = new StaticValidator(c.Name);
                    currentPage.Validators.Add(v);
                    tempResult = false;
                }
            }
        }
    }
}

```

```

        }
    }
    return tempResult;
}
}
}

```

Validating input

The EcoDataSource has two properties of relevance `ValidateRowOnInsert` and `ValidateRowOnUpdate`. If these are `True` the EcoDataSource's `ValidatingRow` event will be triggered whenever a new object is inserted or updated using the EcoDataSource. This event allows the user to implement code which can determine whether or not the changes are valid and, if not, cancel the update. If the update is canceled an exception will be thrown. Validation will be added for updates only:

```

protected void AuthorDataSource_ValidatingRow(object sender,
Eco.Web.UI.WebControls.EcoDataSourceObjectEventArgs e)
{
    if (!PageHelper.ValidatePage(Page, e.AffectedObject))
        e.Cancel = true;
}

```

An exception being thrown when the user tries to enter invalid data is undesirable. Therefore the EcoDataSource has a method `IsValidNewValues` which does the following

1. Starts an in-memory transaction
2. Applies the new values to the object instance
3. Triggers the `ValidatingRow` event to allow the page to determine whether or not the changes are valid
4. Rolls back the in-memory transaction
5. Returns the result of `ValidatingRow`

When the GridView control is about to update the values of a row it will execute its `RowUpdating` event. This event contains information about the change such as a collection of the new values. Using this information it is possible to ask the EcoDataSource if the new values are valid as described above

```

protected void AuthorGridView_RowUpdating(object sender, GridViewUpdateEventArgs e)
{
    if (!AuthorDataSource.IsValidNewValues(e.Keys, e.NewValues))
        e.Cancel = true;
}

```

If the EcoDataSource says the new values are not valid then the GridView will cancel its update. This prevents the GridView from leaving the Editing mode and more importantly prevents it from attempting to update the EcoDataSource and therefore avoiding the exception.

Testing the validation

Add a `ValidationSummary` control to the form from the tool palette. Then run the application again, edit an author and try to update the row with an invalid email address. Instead of allowing the author to be updated the page will display an error message and prevent the update from taking place.

Welcome to my ECO driven website

♦ Email address required

	<u>Auto publish</u>	<u>Salutation</u>	<u>First name</u>	<u>Last name</u>	<u>Full name</u>	<u>Email address</u>	<u>Password</u>
Edit	Delete <input type="checkbox"/>	Mr	John	Smith	Mr John Smith	me@home.com	secret
Update Cancel Delete <input checked="" type="checkbox"/>		<input type="text" value="Mr"/>	<input type="text" value="Fred"/>	<input type="text" value="Smith"/>	<input type="text" value="Mr Fred Smith"/>	<input type="text"/>	<input type="text" value="secret"/>
<input type="button" value="New Author"/>							

This approach to validation is particularly valuable because object validity is determined by the business model, and therefore the same rules may be applied across all user interface implementations.

If you re-run the application and try to add a new Author by clicking the "New Author" button you will encounter a `RowValidationFailedException`. This is because the `AuthorDataSource.ValidatingRow` event is used for both updates and inserts. When you click the `NewAuthorButton` the method `AuthorDataSource.Insert()` is executed which creates a new instance of the `Author` class, but as its properties are all empty the new object fails validation.

There are a number of ways of dealing with this situation.

1. Set `AuthorDataSource.ValidateRowOnInsert` to false. No validation will be performed when `AuthorDataSource.Insert()` is executed.
 - PRO: Very simple solution.
 - CON: You have to rely on the user click Edit after adding a new Author in order to make the Author contain valid values.
2. Ensure that the `Author` class has suitable default values on its UML attributes.
 - PRO: Another simple solution.
 - CON: The `Author` class requires a unique email address per Author.
 - CON: You cannot default these details with the correct values.
3. Change the `NewAuthorButton.Click` method like so:

```
Author newAuthor = new Author(EcoSpaceManager1.EcoSpace);
newAuthor.AutoPublishArticles = false;
newAuthor.Salutation = "Mr";
newAuthor.FirstName = "Peter";
newAuthor.LastName = "Morris";
newAuthor.EmailAddress = "me@home.com";
newAuthor.Password = "secret";
EcoSpaceManager1.EcoSpace.GetEcoService<IPersistenceService>().UpdateDatabase(newAuthor);
AuthorGridView.DataBind();
```

- PRO: It works.
 - CON: The default values cannot be expected to be correct.
 - CON: It bypasses the validation, so the values you enter might actually be incorrect once the constraints on the `Author` class change.
4. Create a secondary form for creating new Author instances.
 - PRO: The `Author` is created from user entered values.
 - PRO: The object won't be saved unless it passes validation.

- CON: It requires development of an additional form.

6.3 Exercise for the reader

Create a new web form to enable you to add or remove article types (ArticleType class.) Make sure you add validation so that users cannot enter invalid data and confirm deletion when the user clicks the Delete link.

7 Passing objects between page requests

Selecting an item from a list and then redirecting the user to a page that manipulates the selected item in some way is a very common website feature. This next example will show how to select an author on the main page and then show the author's articles. Afterwards another page will be added to deal with adding and editing articles.

Creating a link in the GridView control

1. Edit the columns on AuthorGridView.
2. Add a HyperLink column.
 - Set Text to Articles.
 - Set DataNavigateUrlFields to ID
 - Set DataNavigateUrlFormatString to

```
AuthorArticles.aspx?AuthorID={0}
```

Now run the application again and click the Articles link next to an author, you will see that you are redirected to <http://localhost:8080/AuthorArticles.aspx?Author=1>, which at the moment will display a Page Not Found error.

Creating the target page

1. Create a new Web Content Form AuthorArticles
2. Add an EcoSpaceManager component
 - Set its EcoSpaceTypeName property
3. Add an EcoDataSource component
 - Set its ID to ArticleDataSource
 - Set its EcoSpaceManagerID to EcoSpaceManager1
5. Use the [...] editor on ArticleDataSource's Parameters property
 - Click the Add Parameter button
 - Name the parameter param_AuthorID - you may edit the name by single clicking the parameter name in the grid
 - Set the parameter's Parameter Source to QueryString
 - Set the Query String Field to AuthorID
 - Click the "Show advanced properties" hyperlink to expose more properties
 - Change "Type" to Int32
 - Click OK
6. Set the ArticleDataSource.PsExpression property to the following OCL expression

```
Author.allInstances->select(ID = param_AuthorID).Articles
```

Note: param_AuthorID is case sensitive.

6. Add a GridView to the form
 1. Set its ID to ArticleGridView
 2. Set its DataSource ID to ArticleDataSource
 3. Use the [...] editor on Columns so that you have only the following columns.
 - Title
 - State
 - ArticleType

If you have articles in your database from running your Winforms application you will see them listed on the page. This is how the process works.

Default.aspx

1. When the list of authors is displayed on Authors.aspx it will render a column named "Articles".
2. This column is a HyperLink column, its link is the url defined by DataNavigateUrlFormatString.
3. DataNavigateUrlFormatString is the string to be parsed in order to produce the final URL. To produce the final value it is passed to the .NET String.Format() method, allowing for runtime parameters to be injected into the string using placeholders such as {0}, {1}, {2}.
4. DataNavigateUrlFields contains a list of column names on the GridView's data source (AuthorDataSource). The {0}, {1}, {2} etc are replaced with the value of these columns. In this example {0} is replaced with the ID of the author, giving the valueAuthor=1

AuthorArticles.aspx

1. The page is rendered, this causes the ArticleGridView to render.
2. The ArticleGridView requires data to bind to, so this data is retrieved from its data source (ArticleDataSource).
3. ArticleDataSource sees that it has a Parameter named "param_AuthorID", so it creates an OCL variable with the same name.
4. The ArticleDataSource then sets the value of this variable as instructed; it sets it to the value of the AuthorID value in the URL query string.
5. ArticleDataSource evaluates its PsExpression and Expression in turn.
6. The PsExpression results in all authors with the specified ID (of which there can be only one) and returns that author's articles.

Note: PsExpression was used instead of Expression so that the data filtering is performed in the database instead of loading all Author instances into memory and filtering.

EcoDataSource.Parameters may be used to define OCL variables from a number of sources. This will be demonstrated further when a master detail example is created.

8 Editing a single object

So far all editing has been performed as a list of objects in a GridView control. It is common for websites to allow the editing of single objects on a page, providing a more complex page layout with controls such as embedded word processing for editing article contents for example. This next example will show how to create a web form of this type.

Creating a link to the new page

Add a new column to the ArticleGridView in AuthorArticles.aspx. The column should be a HyperLink column with the following settings

- Set its Text to Edit
- Set its DataNavigateUrlFields to ID
- Set its DataNavigateUrlFormatString to EditArticle.aspx?ArticleID={0}

Creating the target page

1. Add a new Web Content Form named EditArticle
2. Add an EcoSpaceManager and set its EcoSpaceTypeName property
3. Add an EcoDataSource
 - Set ID to "ArticleDataSource"
 - Set its EcoSpaceManagerID
4. Add a new Parameter to the ArticleDataSource
 - Set its Name to param_ArticleID.
 - Set its Parameter Source to QueryString.
 - Set its Query String Field to ArticleID.
 - Click "Show advanced properties".
 - Change "Type" to Int32.
5. Set the ArticleDataSource.PsExpression to

```
Article.allInstances->select(ID = param_ArticleID)
```

6. Evaluations performed in the database using PsExpression must always return a collection of objects, even if there is only one item in the collection. To obtain a reference to only a single article we must add an OCL expression to the Expression property which will then be evaluated against the result of the PsExpression evaluation. Set ArticleDataSource.Expression to

```
self->first
```

Note: "self" (which is case sensitive) refers to the element returned by evaluating PsExpression, which in this case is a Collection(Article).

7. From the Data category in the toolbox add a FormView control to the form.

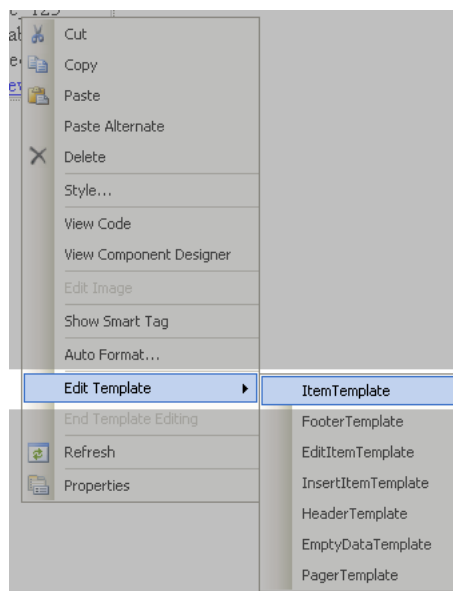
- Set its ID to ArticleFormView
- Set its DataSourceID to ArticleDataSource

Now run the application. Select the Articles link against an author, and then the Edit link against an article, you will now see a read only display of the article selected with Edit, Delete, New links at the bottom.

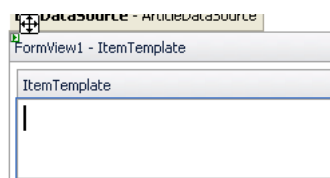
Editing the article

As we intend to have the user edit the article rather than viewing it (a separate ShowArticle.aspx would make more sense in this case anyway) we need to ensure that ArticleFormView goes immediately into Edit mode when the page is shown. In addition we will remove the View template.

1. Right-click ArticleFormView and from the context menu select the Edit Template->Item Template item.



2. The control will now allow you to edit the Item Template, which is the view you see by default. Select all of the controls within the Item Template area and delete it.

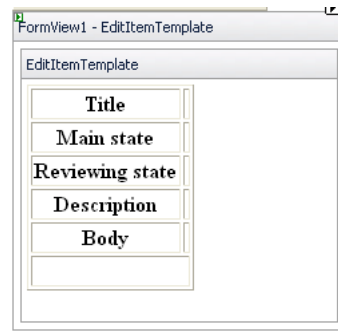


3. Perform the same steps for InsertItemTemplate.

4. Now right-click the control again and select EditItemTemplate from the context menu.

5. Again select all of the controls within the template area and delete them.

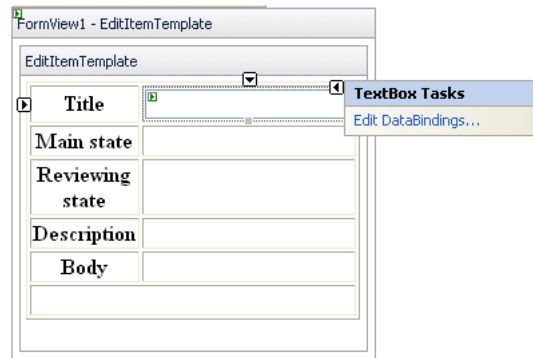
6. Now add a table to the template's client area, edit the table so that it looks something like the following illustration.



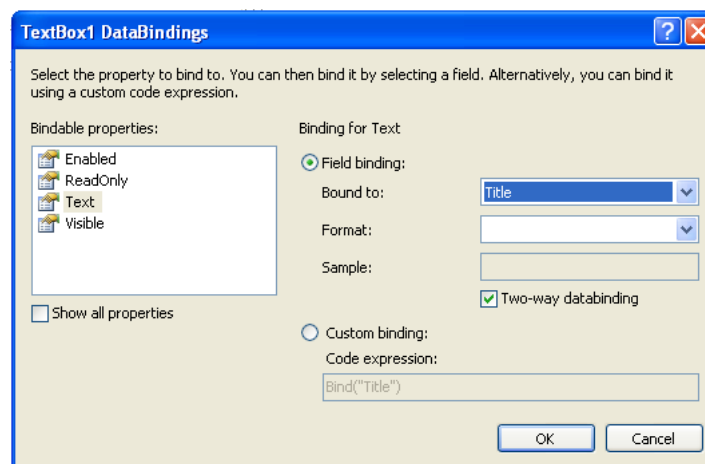
You may wish to copy / paste the following HTML into the ASPX file

```
<EditItemTemplate>
  <table border="1">
    <tbody>
      <tr>
        <th>Title</th>
        <td></td>
      </tr>
      <tr>
        <th>Article type</th>
        <td></td>
      </tr>
      <tr>
        <th>Main state</th>
        <td></td>
      </tr>
      <tr>
        <th>Reviewing state</th>
        <td></td>
      </tr>
      <tr>
        <th>Description</th>
        <td></td>
      </tr>
      <tr>
        <th>Body</th>
        <td></td>
      </tr>
    </tbody>
    <tfoot>
      <tr>
        <td colspan="2"></td>
      </tr>
    </tfoot>
  </table>
</EditItemTemplate>
```

7. From the Web Controls category in the tool palette add a TextBox to the cell to the right of the cell entitled "Title". When the text box is added you will see a Tasks window.



8. Click the Edit DataBindings link.
9. In the form that appears
 - Ensure that the Text property is selected in the list on the left.
 - Select the "Field binding" radio button.
 - Set "Bound to" to Title.
 - Ensure that Two-way databinding is checked.
 - Click OK.



10. Sometimes these radio buttons will not be enabled, if you experience this you should perform the following steps instead.
 - Ensure that the Text property is selected in the list on the left.
 - The Custom binding radio button will be enabled by default.
 - In the Code expression text box enter `Bind("Title")`
 - Click OK.
 11. Set the ID of the text box to TitleTextBox and its MaxLength to 64.
 12. Now add text boxes for Description and Body, ensuring that you set the TextMode properties to MultiLine, and that you set the MaxLength of Description to 255.
- For the State add Label controls instead, and data bind it using the same technique.

Saving or cancelling changes

To add the ability to save or cancel changes two LinkButton controls will be added.

1. Add a LinkButton.
 - Set its ID to UpdateLinkButton
 - Set its Text to [Update]
 - Set its CommandName to Update
2. Add another LinkButton.
 - Set its ID to CancelLinkButton
 - Set its Text to [Cancel]
 - Set its CommandName to Cancel
 - Set its OnClientClick property to

```
return confirm("Are you sure you want cancel your changes?");
```

3. We need to ensure that the form view automatically goes into edit mode when the page is displayed. Select the ArticleFormView control and set its DefaultMode property to Edit.
4. To return to the author's list of articles once the changes have been saved or canceled add the following code to ArticleFormView.ModeChanged:

```
protected void ArticleFormView_ModeChanged(object sender, EventArgs e)
{
    IOclPsService oclPsService =
    EcoSpaceManager1.EcoSpace.GetEcoService<IOclPsService>();
    string ocl = string.Format("Article.allInstances->select (ID={0}).Author",
    Request.QueryString["ArticleId"]);
    IList<Author> Authors = oclPsService.Execute(ocl).GetAsIList<Author>();
    Response.Redirect(string.Format("AuthorArticles.aspx?AuthorID={0}", Authors[0].ID));
}
```

Note that this requires you to add the following name spaces to your "using" clause.

- System.Collections.Generic
- QuickStart.Model
- Eco.Services

Run the application and navigate to an article. You will see that UI is populated with the values of the selected article. You may modify these values and then click either Update or Cancel.

8.1 Databinding a dropdown list

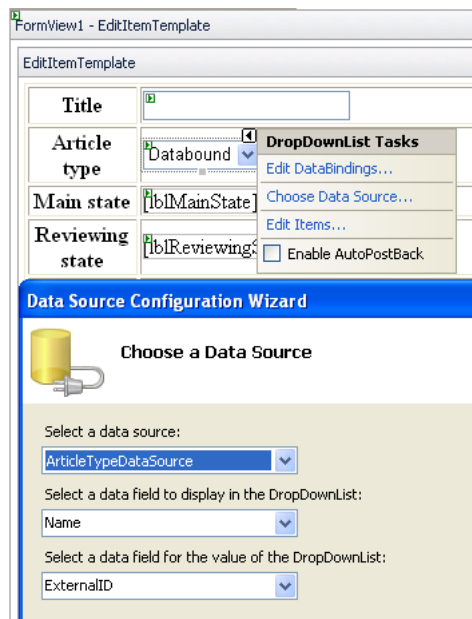
Next a drop down list will be added to the EditArticle form. This drop down list will present the user with a list of all available ArticleType instances from which the user may choose the correct type for the current article. By default the current Article.ArticleType should be selected in the drop down list and when the user clicks the Update link the article should be updated accordingly.

1. Add an EcoDataSource to the form
 - Set its ID to ArticleTypeDataSource
 - Set its EcoSpaceManagerID to EcoSpaceManager1

- Set NullRowMode to First. This will add an empty row at the top of the list of article types so that null values may be represented
- Set its Expression to

```
ArticleType.allInstances->orderBy(name)
```

2. Right-click the FormView control and edit the EditItemTemplate
3. Add a DropDownList to the cell to the right of the label "Article type"
 - On the Tasks window select Choose Data Source



- Set the data source to ArticleTypeDataSource
- Set the display field to Name
- Set the value field to ExternalId
- Set its ID to "ArticleTypeDropDownList"

This will populate the drop down with a list of all article types in the data storage. It will display the Name of each, but the SelectedValue property will return the ExternalId of the object.

Note: The ExternalId is a reference to an instance of a modeled class, presented in the form of a string. The ExternalId is will not change during the life of the web application, however it may change when your model is modified. It is recommended that you do not use ExternalId in URLs which may be spidered by a search engine.

These steps will present a drop down list of all ArticleType instances. The "Name" will appear in the list and the "ExternalID" will be used in the HTML to identify the object's value. Next the drop down will be linked to the current Article's ArticleType property. In addition the Article will be updated with the newly selected value when the Update link is clicked.

Updating the article's property

The web form will use the current Article as the binding context for the drop down, this is because it is parented within a FormView which has its DataSourceID set to ArticleDataSource. As a consequence all that is required is to specify which column the current value should be data bound to.

1. On the Tasks window for the drop down list select Edit DataBindings
2. In the bindings form ensure that SelectedValue is two-way bound to ArticleType

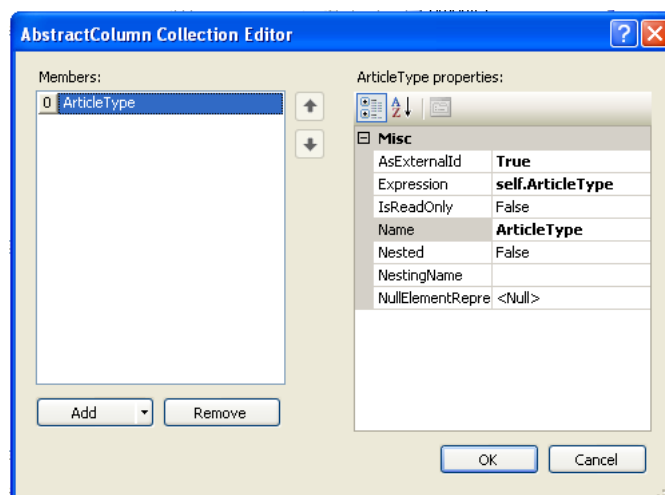
Note: If only the Custom Binding radio button is enabled you should enter `Bind("ArticleType")` as the expression.

There is one final step to perform. When the user interfaces attempts to set the default selected item in the drop down list it will attempt to locate `Article.ArticleType` in a list of ExternalId's, however, columns created automatically by the `EcoDataSource` to represent single-role associations (links to a single ECO object) are added with the following expression

```
self.ArticleType.asString
```

The purpose of this is to enable the object to be displayed in a grid etc as a meaningful piece of text such as "Mr Peter Morris". Unfortunately this is incompatible with what we are currently trying to achieve. Not only do we not want to see the default string representation of the article type, we do not want the object reference either. What we need is a way to read and write the value as an ExternalId string.

1. Use the [...] editor on the ArticleDataSource columns property.
2. In the form that appears click the Add button to add a new column.
3. Name the new column ArticleType
4. Set its expression to `self.ArticleType`
5. Set `AsExternalId` to True



Note: When columns are created automatically at runtime the ArticleDataSource will see that a column named "ArticleType" already exists and will skip creating the column automatically.

Now when the article type drop down list's default value is determined it will read the `ArticleDataSource.ArticleType` column and find a string. This string represents the `ExternalId` of the referenced `ArticleType` (E.g. 12!34). When a new value is selected in the drop down list and the `Update` button is clicked the value of `ArticleDataSource.ArticleType` will be set to `ArticleTypeDropDownList.SelectedValue`, which will be the `ExternalId` for the currently selected article type.

9 Master / detail

This example will demonstrate how to create master / detail relationships. Selecting an Author in a master grid will reveal all related Articles in a second grid.

Adding the master grid

1. Create a new Web Content form to the Admin folder.
2. Name the new item MasterDetail.aspx.
3. Add an EcoSpaceManager and EcoDataSource, and configure them appropriately.
4. Edit the EcoDataSource
 - ID = AuthorDataSource
 - Expression = Author.allInstances
5. Copy the AuthorGridView from Authors.aspx and paste it into the form, make sure you clear out the events for the grid in the object inspector
 - AutoGenerateSelectButton = True
 - AutoGenerateEditButton = False
 - Remove the Delete column
 - Remove the Articles column

Adding the detail grid

Now we have a read only grid showing all existing authors, next a child grid will be displayed showing all articles for an author whenever the Select column next to an author is clicked.

1. Add another EcoDataSource to the form
 - ID = ArticleDataSource.
 - EcoSpaceManagerID = EcoSpaceManager1

Now to make the ArticleDataSource aware that it is a child of the selected author.

2. Use the [...] editor on the Parameters property and add a parameter
 - Name = param_AuthorExternalID - you may single-click the name in the grid to edit it if you wish to change it.
 - Parameter source = Control.
 - Control ID = AuthorGridView
3. Edit the ArticleDataSource's Expression property and enter the following OCL

```
Author.objectfromExternalId(param_AuthorExternalID).articles
```

Note: param_AuthorExternalID is case sensitive.

4. Add a GridView to the form

- ID = ArticleGridView.
 - DataSourceID = ArticleDataSource
 - Edit the columns and remove all columns except for the following
 - Title
 - State
 - ArticleType
5. Add the following implementation for the AuthorGridView's SelectedIndexChanged event

```
protected void AuthorGridView_SelectedIndexChanged(object sender, EventArgs e)
{
    DataBind();
}
```

Now run the application. You should now have a working Master / Detail page.

Welcome to my ECO driven website

<u>Auto publish</u>	<u>Salutation</u>	<u>First name</u>	<u>Last name</u>	<u>Full name</u>	<u>Email address</u>	<u>Password</u>
<u>Select</u> <input type="checkbox"/>	Mr	John	Smith	Mr John Smith	me@home.com	secret
<u>Select</u> <input checked="" type="checkbox"/>	Mr	Fred	Smith	Mr Fred Smith	him@home.com	secret

<u>Title</u>	<u>MainState</u>	<u>ReviewState</u>	<u>ArticleType</u>
My first article	Reviewing	CheckingSpelling	C#
My second article	Authoring		WinForms

10 Authorization

Restricting access to certain parts of your website required two things, authentication and authorization. Authentication is the stage where the user identifies themselves, authorization is the stage that occurs before every request which uses the information from the authentication stage to determine whether or not to permit access to the requested resource.

Implementing authentication / authorization is really no different in an ECO driven website. This part of the document is included partly to demonstrate how to consume some ECO services in code rather than depending on the EcoSpaceManager and EcoDataSource components.

1. First the MasterPage for the website needs a ValidationSummary control on it. If you have one on any of your other forms remove them.
 2. On the Author class add a new UML Attribute
 - Name = IsAdministrator
 - Type = bool
 - Initial value = false
 3. Generate source code for the model.
 4. Rebuild the solution.
 5. Use the Persistence Mapper Provider to evolve the DB if not using XML as your persistence.
- A change is now required to allow you to specify that some Authors are administrators.

6. Open the designer for Authors.aspx.
 7. Edit the Columns on AuthorGridView and add a CheckBoxField databound to IsAdministrator.
 8. Run the website and ensure at least one of your Author instances is marked as an administrator.
- Next the Web.Config file will be changed to enable Forms authentication.

9. Edit the Web.Config file.
10. Locate the <authentication> node and change it as follows:

```
<authentication mode="Forms">
  <forms
    name="ECOQuickStartCookie"
    loginUrl="/Login.aspx"
    protection="All"
    timeout="180"
  />
</authentication>
```

This enables Forms authentication, and additionally specifies the URL to the login page. Next the Login page will be created.

11. Create a new Web Content page.
12. Name it Login.aspx.
13. Add a TextBox and name it EmailAddressTextBox.

14. Add a TextBox and name it PasswordTextBox.

15. Add a Button and name it LoginButton.

```
<table>
  <tbody>
    <tr>
      <th>Email address</th>
      <td>
        <asp:TextBox ID="EmailAddressTextBox" runat="server"></asp:TextBox>
        <asp:RequiredFieldValidator ID="EmailAddressRequiredValidator"
runat="server"
        ControlToValidate="EmailAddressTextBox" ErrorMessage="Email address
required">Required</asp:RequiredFieldValidator>
      </td>
    </tr>
    <tr>
      <th>Password</th>
      <td>
        <asp:TextBox ID="PasswordTextBox" runat="server"
TextMode="Password"></asp:TextBox>
        <asp:RequiredFieldValidator ID="PasswordRequiredValidator" runat="server"
        ControlToValidate="PasswordTextBox" ErrorMessage="Password
required">Required</asp:RequiredFieldValidator>
      </td>
    </tr>
  </tbody>
  <tfoot>
    <tr>
      <td colspan="2" />
      <asp:Button ID="LoginButton" runat="server" onclick="LoginButton_Click"
Text="Login" />
    </tr>
  </tfoot>
</table>
```

Now the HTML has been added to allow the user to input their email address and password the ECO components will be added to allow these credentials to be checked.

16. Add an EcoSpaceManager to the form, and set its EcoSpaceTypeName property.

17. Add an EcoDataSource component to the form

- Name = AuthorDataSource
- EcoSpaceManagerID = EcoSpaceManager1

18. Edit the Parameters for AuthorDataSource.

19. Add a new parameter

- Name = param_EmailAddress
- Parameter source = Control
- Control ID = EmailAddressTextBox

20. Add another new parameter

- Name = param_Password
- Parameter source = Control
- Control ID = PasswordTextBox

Now some code-behind is required to look up the Author with the given email address and password.

```
protected void LoginButton_Click(object sender, EventArgs e)
{
```

```

//Only continue if the ASP validators permit it
if (Page.IsValid)
{
    //Iterate through all Authors returned by the EcoDataSource
    //there will be zero or one only
    Author author = null;
    foreach (IElementProvider currentElementProvider in AuthorDataSource.Select())
        author = currentElementProvider.Element.GetValue<Author>();

    if (author == null)
    {
        //Report the fact that the authentication failed
        Page.Validators.Add(new StaticValidator("Invalid email address or
password"));
        Page.Validate();
    }
    else
    {
        //Get the ExternalID service
        IExternalIdService externalIdService =
EcoSpaceManager1.EcoSpace.GetEcoService<IExternalIdService>();
        //Get the ExternalID for the author
        string currentUserExternalId = externalIdService.IdForObject(author);
        //Store it in the session
        Session["CurrentUserExternalID"] = currentUserExternalId;
        //Notify ASP that the login was successful, and give the author ExternalID
        as the user ID
        FormsAuthentication.RedirectFromLoginPage(currentUserExternalId, false);
    }
}
}

```

The ExternalID (Author PrimaryKey) is used as the user's identity, and this ID is also stored in the Session so that pages can obtain a reference to the user. At the moment there is no protected content on the website.

Restructuring the website

1. Create a new folder named "Admin".
2. Move the following web forms into this folder
 - AuthorArticles.aspx
 - Authors.aspx
 - AutoForm.aspx
 - MasterDetail.aspx
3. Create a new folder named "Members".
4. Move the following web forms into this folder
 - EditArticle.aspx
5. Change the markup for AuthorArticles.aspx so that the URL reference to EditArticle now has the path /Members/EditArticle.aspx.
6. Change the code-behind of EditArticle.cs so that the URL redirected to has the /Admin/ path at the beginning.

Adding authorization

Next the pages within the Members and Admin folders will be protected from anyone who is not signed in.

1. Edit the Web.Config file, beneath the <configuration> node add the following

```
<location path="Members">
  <system.web>
    <authorization>
      <deny users="?"/>
    </authorization>
  </system.web>
</location>
<location path="Admin">
  <system.web>
    <authorization>
      <deny users="?"/>
    </authorization>
  </system.web>
</location>
```

2. Set /Admin/Authors.aspx as your website's default start page.
3. Run the application.
4. You will now be redirected to the Login page.
5. Enter an invalid email / password and you will see the error message telling you that your information is incorrect.
6. Enter a valid email / password and you will be redirected to the Authors.aspx page.

10.1 Role based authorization

The authorization so far will allow any user access to the Admin folder. Next access to this folder will be restricted to those Author's who have IsAdministrator set to True.

1. Add an attribute to the Author class: IsAdministrator: Boolean
2. Change the Web.config file so that the authorization section for the Admin folder reads as follows

```
<location path="Admin">
  <system.web>
    <authorization>
      <allow roles="Admin"/>
      <deny users="*" />
    </authorization>
  </system.web>
</location>
```

Note: You must allow the role "Admin" before denying access to all users "*".

If you now try running the application you will see that you cannot access the Admin folder at all. To define which roles the current user has a change is required in the Global.asax file.

2. Edit the Global.asax file and add the following method

```
void Application_AuthenticateRequest(object sender, EventArgs e)
{
```

```

    if (HttpContext.Current.User != null)
    {
        //Only support Forms authentication
        if (HttpContext.Current.User.Identity.AuthenticationType != "Forms")
            throw new Exception("Only forms authentication is supported, not " +
                HttpContext.Current.User.Identity.AuthenticationType);

        //Get the ExternalID of the current user (if any)
        System.Security.Principal.IIdentity userId =
            HttpContext.Current.User.Identity;

        //Do we have some roles to retrieve? If so, replace the user object
        if (userId.Name != null)
        {
            //Get a new EcoSpace
            QuickStart.QuickStartEcoSpace ecoSpace = new QuickStart.QuickStartEcoSpace();
            ecoSpace.Active = true;

            //Get the External ID Service
            Eco.Services.IExternalIdService externalIdService =
                ecoSpace.GetEcoService<Eco.Services.IExternalIdService>();

            //Get the author
            QuickStart.Author author =
                externalIdService.ObjectForId(userId.Name).GetValue<QuickStart.Author>();
            if (author != null)
            {
                System.Collections.Generic.List<string> roles = new
                System.Collections.Generic.List<string>();
                roles.Add("Member");
                if (author.IsAdministrator)
                    roles.Add("Admin");

                //Replace the user with our new user, including their roles.
                HttpContext.Current.User = new
                System.Security.Principal.GenericPrincipal(userId, roles.ToArray());
            }
        }
    } //user != null
}

```

3. Add a new Default.aspx Web Content form to the root of your project with the two following links

```

<a href="/Members/MyArticles.aspx">My articles</a>
<a href="/Admin/Authors.aspx">Authors</a>

```

4. Re-run the application and click the MyArticles link (this page does not yet exist).
5. You will be presented with a login page, sign in as the non-administrator Author.
6. You will not be permitted to access /Members/MyArticles.aspx which will show you a not-found error.
7. Now go back to /Default.aspx and click the Authors link.
8. You will be directed to the Login page again, this is because you need to log in as an administrator to see this page.

10.2 Accessing the current user

Previously you created a Default.aspx page with a link to a page /Members/MyArticles.aspx which did not exist. This page will be created next in order to demonstrate how to access the user currently logged in.

1. Create a new Web Content form in the Members folder named MyArticles.aspx.
2. Add an EcoSpaceManager and EcoDataSource and configure them appropriately.
3. Name the EcoDataSource "ArticleDataSource".
4. Edit the Parameters of ArticleDataSource and add a parameter
 - Parameter source = Session
 - Session field = CurrentUserExternalID
5. Set the Expression property to the following OCL expression

```
Author.objectfromExternalId(param_CurrentUserExternalID).Articles
```

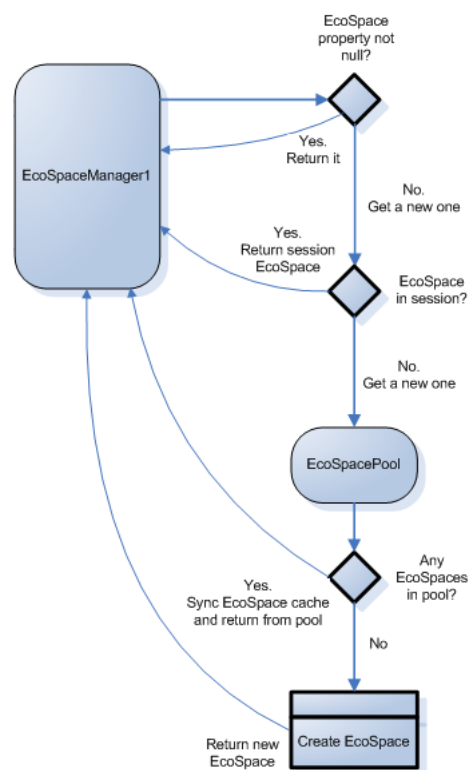
6. Add a GridView (ArticlesGridView) to the form and connect it to the AuthorDataSource component.
7. Add a HyperLinkField column to the grid
 - Text = Edit
 - DataNavigateUrlFields = ID
 - DataNavigateUrlFormatString = EditArticle.aspx?ArticleID={0}

If you now re-run the application and click the "My articles" link on the main page you will be prompted to sign in. Once signed in you will be presented with a form showing the articles of the Author who's credentials you just used to sign in with.

11 EcoSpace retrieval strategies

The EcoSpaceManager component has been used quite extensively throughout this document. This component is responsible for providing an EcoSpace of the requested type, but so far no mention has been made regarding where this instance comes from. It would be reasonable to assume that the EcoSpaceManager merely creates a new instance for each page request. This is in fact accurate when describing the default behavior, but it is possible to modify this default behavior in order to fine tune your requirements between speed and memory usage.

The following image is a subset of the one presented in the section Logical flow (see page 10) near the beginning of this article.



As you can see there is an entity entitled "EcoSpacePool". This allows the EcoSpaceManager to retrieve an existing EcoSpace instance from a pool, and also to return it when the page has finished with it. Retrieving an EcoSpace from a pool improves performance because the most commonly used object instances are likely to have already been loaded into the cache, avoiding a trip to the data storage altogether.

11.1 Pooling EcoSpaces

The settings for pooling are defined in the Web.config file

```
<appSettings>
  <add key = "Eco.Web.MaxPool" value = "0" />
  <add key = "Eco.Web.MaxAge" value = "600" />
  <!-- Non relevant settings omitted -->
</appSettings>
```

Eco.Web.MaxPool specifies how many instances may be present in the pool. The default value of zero indicates EcoSpace pooling should not be used. Changing this value to anything greater than zero will enable pooling.

Eco.Web.MaxAge specifies the maximum age of an EcoSpace in the pool. The purpose of the pool is to provide a pool of EcoSpace instances with commonly accessed data in them. An EcoSpace will be retrieved from the pool, used, and returned many times during its lifetime. It is therefore probable that during its lifetime it also loads object instances which will not be used very often, or even used at all. In order to prevent the EcoSpaces' caches from becoming "bloated" this setting allows you to determine the maximum age of an EcoSpace in the pool. Once an EcoSpace is older than the specified amount of time it will be discarded and no longer be available within the pool.

Here is an outline of the pooling process:

Retrieve from pool

1. Is there an EcoSpace in pool?
 - No: Return a new instance.
 - Yes: Is the age of the EcoSpace less than the MaxAge?
 - Yes: Return it.
 - No: Dispose of it, and repeat step 1
2. Invalidate any parts of the cache that other users have made changes to (multi-user synchronization will be covered in a later document).

Return to pool

1. Is the pool full?
 - Yes: Dispose of the EcoSpace.
 - No: Is the EcoSpace too old to be returned?
 - Yes: Dispose of the EcoSpace.
 - No: Return it to the pool.

11.2 Session EcoSpaces

When returning an EcoSpace to the pool there must be no dirty objects (objects with changes that need saving). If this restriction was not in place it would be possible to retrieve an EcoSpace from the pool containing changes made by another user, this behavior would be confusing and would also be likely to introduce errors.

The EcoSpaceManager allows the developer to specify an additional option for the retrieval / return of EcoSpace instances, this option is specified via its SessionStateMode property.

Value	Setting
Never	The EcoSpaceManager will not use the user's Session when retrieving or returning an EcoSpace instance.
IfDirty	<p>If there are dirty objects in the EcoSpace at the end of the page request the EcoSpace will be stored in the user's Session using the key given in the SessionStateKey property.</p> <p>When an EcoSpace instance is required the EcoSpaceManager will first inspect the user's Session using the given key, if an EcoSpace is stored within the Session under this SessionStateKey then that instance will be used.</p>
Always	<p>This option is essentially the same as IfDirty, except the EcoSpace will always be stored in the user's Session at the end of the page request whether it has dirty objects or not.</p> <p>This option is useful when you wish to have instances of Transient classes in your EcoSpace. A Transient class can never be considered "dirty" because it is never updated to the data storage.</p>

As it is possible to use more than one EcoSpaceManager on a single page request it is possible for a page to use multiple instances of EcoSpaces. Some benefits of this are

- The ability to utilize more than one type of EcoSpace and therefore different models.
- To allow updates to objects over multiple pages without having to commit to the data storage. For example a multi-page wizard for creating a new customer account.
- Using different SessionStateKey values in order to logically separate sets of updates. For example you might use "NewCustomer" as the SessionStateKey for creating a new customer over multiple pages, and use "ShoppingCart" is the SessionStateKey for an EcoSpace which holds a single unsaved purchase order which the user is constantly modifying throughout their visit to the website.

12 Shopping cart

The "Shopping cart" example will be demonstrated next. Some modifications will be made to the model to allow an Author to purchase the right to view other authors' articles. The order will remain in an unsaved state across multiple page requests by storing it in the user's Session state. Finally, if the user chooses to, the order will be saved to the data storage and they will be given access to the articles they have (fictitiously) purchased. The example will not go as far as implementing all of the requirements such as only giving access to purchased articles etc as that would distract from the true purpose of this example, which is to demonstrate how to perform the following tasks

1. Modify an object across multiple page requests.
2. Perform these changes in isolation, so that the user may switch to another task part way through and then continue without any adverse affects.

12.1 Preliminary changes

Before implementing the cart there are a number of changes to be made to the existing features of the site.

Giving articles a price

1. Select the Article class in the modeler.
2. Add a new UML attribute
 - Name = Price
 - Type = decimal
 - Initial value = 0
 - Default DB value = 0
3. Add a constraint to the Article class
 - Name = Price must be at least 1 if not free
 - Expression = (price = 0) or (price >= 1)
4. Save changes
5. Generate source code
6. Rebuild the project
7. Evolve the DB if not using the XML persistence component.

Next the EditArticle form will be updated to allow for the input of a price.

7. View the markup for EditArticle.aspx
8. Beneath the markup for ArticleType add the following

```
<tr>
  <th>Price</th>
  <td>
    <asp:TextBox ID="PriceTextBox" runat="server" Columns="8" MaxLength="8"
      Text='<%# Bind("Price") %>'></asp:TextBox>
```

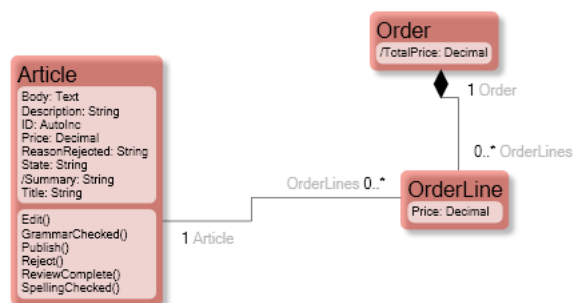
```
</td>
</tr>
```

9. Run the application and enter some prices for the articles.

12.2 Adding orders to the model

Next some classes will be added to the model to allow the website to record order information.

1. Add a new Class Diagram to the model and name it "Shopping cart"
2. Drag the Article class from the treeview in the Modlr-tab onto the new diagram
3. Add two new classes and name them "Order" and "OrderLine" respectively.
4. Draw a class association line from Order to OrderLine
 -
5. Draw a class association line from Article to OrderLine
 - Adjust the multiplicities and compositions according to the diagram below
6. Add a new attribute to the OrderLine class
 - Name = Price
 - Type = decimal
7. Add a new attribute to the Order class
 - Name = TotalValue
 - Type = decimal
 - Attribute type = Derived (read only)
 - Derivation OCL = self.OrderLines.Price->sum
8. Save changes to the model
9. Generate code
10. Rebuild the solution
11. If you are not using XML persistence click the "Evolve DB" button on the persistence mapper provider



Next some additions will be made to the model source code to make the process easier.

1. Add the following method to the Article class (in Article.cs):

```
public static Article GetByID(IEcoServiceProvider serviceProvider, int id)
{
    //Get the article by its ID
    string query = string.Format("Article.allInstances->select(id = {0})", id);
    IOclPsService oclPs = serviceProvider.GetEcoService<IOclPsService>();
    IList articleElements = oclPs.Execute(query);
    if (articleElements.Count > 0)
        return articleElements[0].GetValue<Article>();
    return null;
}
```

2. Add the following methods to the Order class (Order.cs).

```
private OrderLine GetLineByArticle(Article article)
{
    foreach (OrderLine currentOrderLine in OrderLines)
        if (currentOrderLine.Article == article)
            return currentOrderLine;
    return null;
}

public void AddArticle(int id)
{
    Article article = Article.GetByID(AsIObject().ServiceProvider, id);
    if (GetLineByArticle(article) == null)
    {
        OrderLine newLine = new OrderLine(AsIObject().ServiceProvider);
        OrderLines.Add(newLine);
        newLine.Article = article;
        newLine.Price = article.Price;
    }
}
```

12.3 The shopping cart control

A control will now be created to show the articles currently in the user's shopping cart. This control will then be added to the master page so that it appears on every page.

1. Create a new folder named "Controls"
2. Right click the new folder and on the context menu select "Add" and then "New item"
3. Select Web User Control and click OK
4. Give the control the name ShoppingCart.ascx

Next the required ECO components will be added, and the web controls required to display the contents of the cart.

5. Add an EcoSpaceManager to the control

- Set its EcoSpaceTypeName
- SessionStateKey = ShoppingCart
- SessionStateMode = Always
- UsePool = False

6. Add an EcoDataSource to the control

- ID = CartItemsDataSource
- EcoSpaceManagerID = EcoSpaceManager1
- Expression = Order.allLoadedObjects->first.OrderLines

Note: Instead of "allInstances" the OCL command "allLoadedObjects" is used. This will return a list of objects that have already been loaded into the EcoSpace's cache. As the EcoSpace being used will be private to the user and stored in their session it will contain exactly zero or one Order instances.

Now some web controls will be added to show the contents of the cart.

7. Add a Label control to the control and name it CartTotalLabel

8. Add a GridView control to the control

- ID = CartItemsGridView
- DataSourceID = CartItemsDataSource
- Remove all columns except for Article and Price

9. Edit the columns property on CartItemsDataSource

10. Add a new column

- Name = Article
- Expression = self.Article.Summary

Note: The above steps will override the "Article" column which is added by default. Instead of a column with the expression "self.Article.asString" the column will show Article.Summary, which is the title of the article combined with the name of the author.

EcoSpaceManager - EcoSpaceManager1	
EcoDataSource - CartItemsDataSource	
Article	Price
abc_123	0.3
abc_123	0.3
abc_123	0.3
abc_123	0.3
abc_123	0.3

12. To make life a little easier later add the following static method to the control's source. This will give easy access to the current Order instance.

```
public static Order GetCart()
{
    //Create an EcoSpaceStrategyHandler to manage the EcoSpace for us
    //The settings will ensure it is retrieved from the Session if present
    //otherwise a new instance will be created
    EcoSpaceStrategyHandler ecoSpaceProvider =
        new EcoSpaceStrategyHandler(EcoSpaceStrategyHandler.SessionStateMode.Always,
        typeof(QuickStartEcoSpace), "ShoppingCart", false);

    //Get an instance of the EcoSpace
    QuickStartEcoSpace ecoSpace = (QuickStartEcoSpace)ecoSpaceProvider.GetEcoSpace();

    //Get the only loaded Order in the EcoSpace
```

```

    IOclService oclService = ecoSpace.GetEcoService<IOclService>();
    Order result =
    oclService.Evaluate("Order.allLoadedObjects->first").GetValue<Order>();

    //Create a new order if none was found
    if (result == null)
        result = new Order(ecoSpace);

    //Return the EcoSpace to the session
    ecoSpaceProvider.ReturnEcoSpace(ecoSpace);
    return result;
}

```

Note: The `EcoSpaceStrategyHandler` is the class used internally by the `EcoSpaceManager` component. It is important to use this class when storing `EcoSpace` instances in the session as it stores the `EcoSpace` in a way that makes it possible to use SQL Server session state when web farming.

Next the control will be added to the master page.

13. View the markup for the master page.

14. Change the markup like so

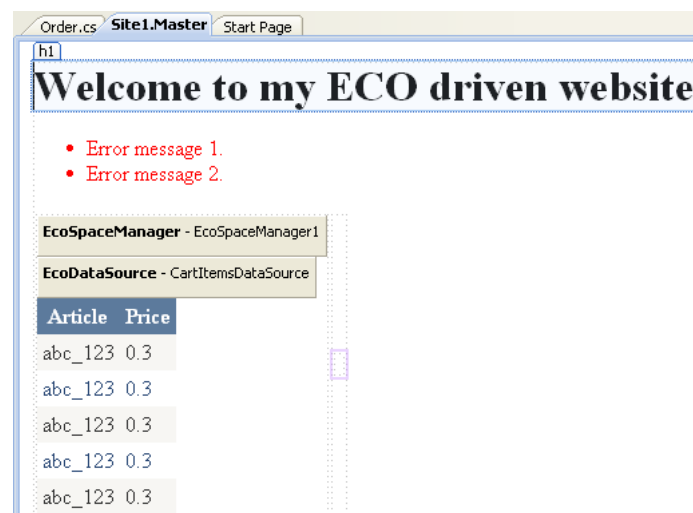
```

<table>
  <tr>
    <td></td>
    <td>
      <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
        </asp:ContentPlaceHolder>
    </td>
  </tr>
</table>

```

Note: If we use tables for layout we will apparently go to hell, therefore I recommend you research CSS layouts.

15. In design mode drag the `ShoppingCart` control and drop it into the empty table cell



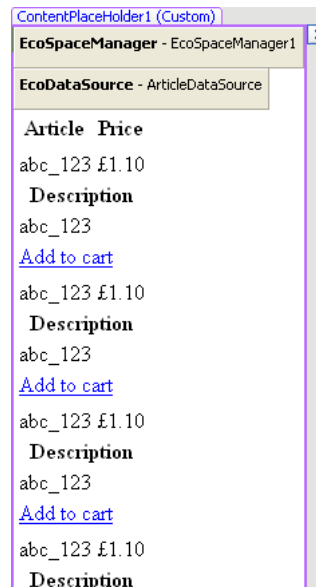
12.4 Show all available articles

Next a web form will be created to show all articles. A link will be added to enable the user to add the article to their shopping cart.

1. Edit Default.aspx and add a hyperlink to /AllArticles.aspx
2. Create a new Web Content Form in the root of the site
3. Name it AllArticles.aspx
4. Add an EcoSpaceManager and set its EcoSpaceTypeName
5. Add an EcoDataSource
 - ID = ArticleDataSource
 - EcoSpaceManagerID = EcoSpaceManager1
 - Expression = Article.allInstances
6. Add a DataList control to the form
 - ID = ArticlesDataList
 - DataSourceID = ArticleDataSource
 - Set its markup as follows

```
<asp:DataList ID="ArticlesDataList" runat="server">
  <HeaderTemplate>
    <table>
      <thead>
        <tr>
          <th>Article</th>
          <th>Price</th>
        </tr>
      </thead>
      <tbody>
    </HeaderTemplate>
    <ItemTemplate>
      <tr>
        <td><%# Eval("Summary") %></td>
        <td><%# Eval("Price", "{0:c}") %></td>
      </tr>
      <tr>
        <th colspan="2">Description</th>
      </tr>
      <tr>
        <td colspan="2"><%# Eval("Description") %></td>
      </tr>
      <tr>
        <td colspan="2">
          <asp:LinkButton ID="AddToCartLinkButton" runat="server"
            CommandName="AddToCart" Text="Add to cart" />
        </td>
      </tr>
    </ItemTemplate>
    <FooterTemplate>
      </tbody>
    </table>
  </FooterTemplate>
</asp:DataList>
```

Note: This markup is a simple repeating pattern with data binding to various properties on Article. A LinkButton has been added to invoke a method to add the Article to the current Order.



7. In the code-behind call `.DataBind()` in the `Page_Load` method.

8. To bind the ID of the article to the LinkButton add the following code to the `ItemDataBound` event.

```
protected void ArticlesDataList_ItemDataBound(object sender, DataListItemEventArgs e)
{
    LinkButton addToCartLinkButton =
    (LinkButton)e.Item.FindControl("AddToCartLinkButton");
    if (addToCartLinkButton != null)
        addToCartLinkButton.CommandArgument = DataBinder.Eval(e.Item.DataItem,
        "ID").ToString();
}
```

9. When the user clicks the "Add to cart" link button the article should be added to the cart. To achieve this add the following implementation to the `ItemCommand` event

```
protected void ArticlesDataList_ItemCommand(object source, DataListCommandEventArgs e)
{
    switch (e.CommandName)
    {
        case "AddToCart":
            int articleID = int.Parse((string)e.CommandArgument,
            NumberFormatInfo.InvariantInfo);
            ShoppingCart.GetCart().AddArticle(articleID);
            DataBind();
            break;

            default:
                throw new NotImplementedException("Unknown command: " + e.CommandName);
    } //switch
}
```

Now run the application, select All Articles and click "Add to cart" next to one or more articles.

Welcome to my ECO driven website

Article	Price	Article	Price
My second article by Mr John Smith	2.5	My second article by Mr John Smith	£2.50
My first article by Mr John Smith	2.5	Description	
Mr first article by Mr Fred Smith	0	My article is written in Latin.	
		Add to cart	
		My first article by Mr John Smith	£2.50
		Description	
		My article is written in Latin.	
		Add to cart	
		Mr first article by Mr Fred Smith	£0.00
		Description	
		This is my article, in Latin.	
		Add to cart	
		My second article by Mr Fred Smith	£0.00
		Description	
		This is my article, in Latin.	
		Add to cart	

13 Summary

This document has demonstrated how to use ECO in an ASP .NET web application. It has demonstrated how to bind an ECO data source to standard web UI controls and explained how EcoSpace lifetime management works. It was not intended to be a comprehensive example of how to implement a fully commercial shopping cart to a website. Instead it is a demonstration of how to modify objects over multiple page requests, and to isolate these operations into separate logical groups so that they do not interfere with each other.

This technique can be used at various places within a large website by using different values for the `SessionStateKey` property on the `EcoSpaceManager` (or when creating an `EcoSpaceStrategyHandler` retrieve the `EcoSpace` instance in code). Although in this example the `SessionStateMode` was set to `Always` you may decide it is more appropriate to set this property to `IfDirty` and allow the `EcoSpace` to cooperate in the pool when there are no unsaved changes.

This document has not covered the use of the ECO web providers (membership, roles etc) as its purpose was to build an application that has a business model which may be used in both a web and WinForm application. Use of these providers may be covered in a later document.

Index

A

Accessing the current user 37
Adding orders to the model 42
Authorization 32

B

Before you start... 3

C

Confirming deletion 13
Creating a web form 12

D

Databinding a dropdown list 26

E

EcoSpace retrieval strategies 38
Editing a single object 22
Exercise for the reader 19

L

Logical flow 10

M

Master / detail 30

N

New ECO ASP.NET components 8

O

Overview 1

P

Passing objects between page requests 20
Pooling EcoSpaces 38
Preliminary changes 41
Prerequisites and goals 2

R

Role based authorization 35

S

Session EcoSpaces 39
Shopping cart 41
Show all available articles 46
State machine operation 6
Summary 49

T

The shopping cart control 43

V

Validating user input 15

W

Web auto forms 4